January 2020

System and Architecture of a Quantum Key Distribution (QKD) Service over SDN

Santanu Ganguly

System and Architecture of a Quantum Key Distribution (QKD) Service over SDN

Author: Santanu Ganguly

Contents

1 ABSTRACT

Presented herein is a system for Quantum Key Distribution Service (QKDServ) architecture that utilizes the novel concept of "Quantum Key on demand" (QKoD) to achieve efficient key resource usage in combination with a new concept of quantum key management server (QKMS) technique, where efficient QKoD relies on the QKMS to allocate secret keys for security requirements in a timely manner on demand. This proposed system also provides a novel QKD over SDN architecture where QKMSs are constructed to secure control channels (C-Ch) and data channels (D-Ch). Finally, the techniques presented herein propose a novel dynamic routing, wavelength and key assignment (QRKWa) algorithm designed to implement QKoD jointly for C-Chs and D-Chs, where the adaptive key assignment strategy considers two cases (i.e., key-updating based on time complexity of attacks and key-updating based on data complexity of attacks).

2 TECHNICAL FIELD

The following relates generally to Quantum Key Distribution (QKD) and more specifically to architecture for QKD management and routing distribution.

3 BACKGROUND

The main cryptographic protocols used today to secure the Internet and the vital services accessed across the Internet, such financial transactions, are all susceptible to attack by the development of a sufficiently large and efficient quantum computer.

In case of classical computation, the complexity of searching in an unstructured data set of size n is $O(n)$, e.g., in the worst possible case, all the records will need to be inspected. The well-known Grover's algorithm allows to solve this problem in $O(\sqrt{n})$, steps. This essentially means that, if we have 40 bits and we need to find a combination that satisfies certain condition, then in the classical case we need to process approximately 10^{12} different combinations, whereas the quantum algorithm will yield a result in about 10^{6}queries.

Rather than depending on the complexity of factoring large numbers, quantum cryptography is based on the fundamental and unchanging principles of quantum mechanics, which in turn the efficient Shor's algorithm is based. The following proposal is for replacing the current day public key cryptography with a variety of quantum algorithm.

Several industry leaders and financial institutions have started testing quantum computing as an option for their future production network. This proposed architecture is that of an implementation of QKD as an integrated service in SDN environment utilizing extended OpenFlow 1.4.

Key generation: QKD involves the process of generating a unique code key that is shared by, and known only to, the sender and recipient. Encryption of the message with a quantum key protects its confidentiality by transforming it into an unintelligible form.

A secure code for the QKD process can be generated in various ways. One such is via an One Time Pad (OTP) device, while another way to generate such a code is via a Quantum random number generator (QRNG). A QRNG is either a device which provide a stream of random bits generated using methods based on the laws of quantum physics or a software implementation of an algorithm which uses qubits in the backend to generate the random numbers making them *truly* random as opposed to the pseudo random numbers generated today in RSA type implementations.

In order to generate true random numbers at a chip level hardware, photons are sent one at a time into a semi-transparent mirror and detected after which, the exclusive event of reflection or transmission are given "0" or "1" values. This QRNG is intrinsically and provably random providing instantaneous and inexhaustible entropy. ID Quantique (EP 2 940923 A1) uses similar elementary quantum optics process to generate true random numbers leveraging photons and have widely accepted industry adoption in addition to being discussed to be standardized at the ITU's Telecommunication Standardization Section (ITU-T).

Currently various prior art options exist for key generation, via both software based and hardware based solutions:

1) Software based options:
    a) D-Wave Systems' digital ocean library with simulated annealing – a code exists today which uses simulated annealing (i.e., code that runs on one's computer) to achieve true random quantum number generation.
    b) Using IBM's Qiskit or Google's Cirq opensource Python libraries, a random number generator can be designed using Hadamard gates and respecting Shannon's Laws for Quantum Cryptography.
    c) An example of a random number generator using 4 bits of IBM Q Experience is shown in Fig. 7a and corresponding pseudo-code in Fig. 7b. Here "H" represents Hadamard gates, "q" represents qubits, "c " represents the control bits.

2) Hardware based options:
    a. ID Quantique: Quantis is a quantum true random number generator developed by ID Quantique. It can be used as a USB devices, PCI or PIC express card.

Once two Quantum Nodes (QN) establish their respective secure keys and authenticates on the QKD network, they can engage in a "BB84 Protocol" over the Quantum Channel (Q-Ch). Any pair of QKD servers can perform a QKD protocol such as BB84. As of recent times, a key management server architecture facilitated by QKarD's (quantum smartcard which are fiber-coupled devices) prototyped by Los Alamos driven research has been demonstrated to perform BB84 over a distance of 50 km using 1550 nm laser. The protocol is shown in Fig. 8.

The first stage of the BB84 protocol is the quantum transmission. For example, assume Alice is on the transmitter and Bob on the receiver site of the quantum channel. Alice randomly chooses two strings independent of each other with length *m*. The first string represents the basis for the quantum transmission and the second the proper value of the specific bit. Alice and Bob have the same mapping scheme in common.

Alice starts to transmit. Therefore, she takes the first bit of the first string, indicating which base to use, and the first bit of the second string indicating which value to take. She applies this procedure on all *m*-bits of both strings and sends them as optical signal via the quantum channel to Bob.

Bob, on his part, also chooses a random string with length *m*, for his base choices. With these bits he measures the incoming signal with the corresponding filter. Bob keeps the measurement results to himself. A consideration of note is that Bob's measurements do not completely match with Alice's measurements. This mismatch may be due to optical misalignment, disturbance on the quantum channel, noise in Bob's detectors, or the presence of an eavesdropper Eve, although both choose the same basis.

The throughput of the quantum channel is given by:

$$g_q = \mu . \alpha_f . \alpha_e . \eta_{det} . k_{dead}$$

where gain of the quantum channel g_q consists now of the mean photon number μ, factor α_f represents the fiber loss (which is distance dependent and increases with higher distances), η_{det} is the receiver's detection efficiency and k_{dead} is a factor accounting for the reduction of the photon detection rate due to the dead time, which is the hold-off time following each detection event; during this time the bias voltage of the device is below a certain level such that no photon can be detected, and α_e is the additional loss of the system.

After the communication between Alice and Bob on the quantum channel is finished, they switch to the *public* channel. At this point, Alice holds the two strings of length *m* containing her choice of base and value and Bob a string of length *m* containing his choice of bases and his measurement results of length $g_q . m$.

4 SUMMARY

In one aspect an architecture for implementing QKD as an integrated service in SDN environment is provided utilizing extended OpenFlow 1.4. Accordingly, QKMSs are constructed over the software-defined optical networks to satisfy these requirements utilizing OpenFlow 1.4. The synchronized secret keys can be stored in the corresponding quantum key distribution server (QKDS) which is embedded in each node. To enhance the secret-key management, the secret keys between each pair of quantum nodes (QN) can be virtualized into a QKMS. QKMS can dynamically provide different number of secret keys for data encryption according to different security requirements between the two nodes. This function relies on the QKMS to allocate secret keys for security requirements in a timely manner on demand and is defined as Quantum Key on Demand (QKoD).

In another aspect, a system to manage QKD over SDN is provided.

5 DESCRIPTION OF DRAWINGS

The features of the invention will become more apparent in the following detailed description in which reference is made to the appended drawings wherein:

Fig. 1 is an architecture diagram for the QKD Service Systems (QKDServ) based on novel OpenFlow 1.4 extensions;
Fig. 2a illustrates a QKDServ key distribution process between two Quantum Nodes (QNs);
Fig. 2b illustrates process of assignment of quantum secure keys in the SDN controlled QKD environment;
Figs. 3a, 3b, 3c, 3d, 3e, and 3f depict various proposed extensions in OpenFlow 1.4 protocol headers which leverages quantum metadata;

Fig. 4 shows the intercommunication workflow among the three architectural planes of the QKDServ for successful QKD service creation;
Fig. 5 is a flowchart for strategies for routing and Quantum Key Rate (QKR) assignment;
Fig. 6a shows Table 1 depicting the algorithm for QKD Routing, Key and Wavelength assignment (QEKWa);
Fig. 6b shows Table 2 which illustrates the notation and definitions of terminology and parameters in the QRKWa algorithm in Table 1;
Fig. 7a is an illustration of a prior art showing a circuit used to generate a quantum random number represented by 4 qubits using IBM's Qiskit and gate model;
Fig. 7b shows a quantum random number generation algorithm;
Fig. 8 shows interaction and communication workflow in a prior art depicting quantum BB84 protocol.

Like reference symbols and numbers in the drawings indicate like elements.

6 DETAILED DESCRIPTION

Embodiments will now be described with reference to the figures. For simplicity and clarity of illustration, where considered appropriate, reference numerals may be repeated among the figures to indicate corresponding or analogous elements. In addition, various specific details are set forth in order to provide a thorough understanding of the embodiments described herein. It will be understood by those of ordinary skill in the art that the embodiments described herein may be practiced without these specific details. In other instances, well-known methodologies, procedures and components have not been described in detail so as not to obscure the embodiments described herein.

Also, the description is not to be considered as limiting the scope of the embodiments described herein. It will also be appreciated that any module, unit, component, server, computer, terminal or device exemplified herein that executes instructions may include or otherwise have access to computer readable media such as storage media, computer storage media, or data storage devices (removable and/or non-removable). Such as, for example, magnetic disks, optical disks, or tape. Computer storage media may include volatile and non-volatile, removable and non-removable media implemented in any method or technology for storage of information, such as computer readable instructions, data structures, program modules, or other data. Examples of computer storage media include RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by an application, module, or both. Any such computer storage media may be part of the device or accessible or connectable thereto. Any application or module herein described may be implemented using computer readable/executable instructions that may be stored or otherwise held by such computer readable media.

1

Quantum cryptographic related capabilities have been recently highlighted as an important functionality for military and federal governmental tactical networks, where assured networking is critical and flexible management are needed – SDN related technology excels at this today. This

versatile and programmable network architecture is expected to be more suitable to support future heterogeneous systems and implement ad-hoc network policies that may arise, for example, in tactical environments. By standards of fast emerging technologies, this needs to include quantum encryption and communication models to secure, encode and transmit information. Quantum Key Distribution (QKD) is a process that uses a quantum channel (Q-Ch) together with a public channel (P-Ch) to securely exchange secret keys between two legitimate parties.

Quantum Key Distribution (QKD) is a process that uses a quantum channel (Q-Ch) together with a public channel (P-Ch) to securely exchange secret keys between two legitimate parties. A Q-Ch is capable of carrying the quantum bits (qubits) which is generated by encoding classical bits into quantum states of light, while a P-Ch is capable of carrying classical data to compare the measurements associated with these qubits as well as perform post-processing correct distillation of the secret keys.

Fig 1. Depicts the QKDServ Systems Architecture (QKD Service Systems) based on novel OpenFlow 1.4 extensions which will be addressed in this document later.

In order to deploy QKDServ for multiple users, an SDN architecture for QKD network control and management is introduced in Fig. 1. This QKDServ architecture framework consists of three planes in a top-down design: Application (100), Control (110) and Infrastructure (130).

Protocols such as OpenFlow 1.4.0 and NETCONF are applied to the southbound interface (109) between infrastructure plane (130) and control plane (110). This is used to transmit control and configuration (406) messages such as control and configure secret-key exchange, storage, relay, assignment, destruction etc.

OpenFlow 1.4.0 (OFP) has been chosen in this architecture as the southbound interface protocol given by 109. Feasibility of using OFP in an SDN-enabled QKD network has been experimentally verified.

In this architectural framework, OpenFlow plugin is extended to implement the southbound interface (109) via a service abstraction layer (105). The northbound interface (103) between control and application planes is implemented with RESTful API and is used to transmit QKD service request/transmit messages.

The request/response messages of QKD service creation (104), modification (106) and deletion (108) are implemented using the POST, PUT and DELETE methods of Hypertext Transfer Protocol (HTTP), respectively.

The timely *key synchronisation and Service configuration* is achieved by a match/action pair flow rule in OpenFlow. This rule modifies and identifies the incoming packets in accordance with the device's flow table.

The same match/action pair flow is used for encrypting specific incoming packets to identify (i.e. match) traffic and apply encryption to it (i.e. action). As OpenFlow support traffic matching down the layer stack, there is no necessity to modify or extend the matching process.

The three planes for this QKDServ architecture are implemented as follows:

1) Application plane (100): This design considers only one QKD network management (102) operator or service provider (QKDServ provider) over the QKD network infrastructure.

The QKDServ provider at the Application plane (100) receives the QKR (Quantum Key Rate) requirements (given by 205 and 207 in Fig. 2a) of each user (101), generates and sends multiple QKD services requests (104) to the control plane (110).
Each QKDServ request includes a specific QKR requirement between two QN's (QKD Nodes) given by 134. A QKR request can need one or more QKR slots (Fig 2a).
The secret-key size can be managed since only one encryption algorithm (One Time Pad or Quantum Random Number Generator) is used for the QKDServ architecture framework.

2) Control Plane (110): The SDN controller (111) is deployed in the control plane (110). This controls and manages the QNs (134) in the infrastructure plane via OFP and OpenFlow agents.
In Fig. 1, the SDN controller interfaces with a database and Network Services (114). The QKR information about the QKD service requests are held in the database.
The SDN controller can be any device which supports OpenFlow.
The controller platform (111) will develop QKDServ functions (112) supporting at least six different modules as follows:
    - Topology (113): used to collect, store and update the QKD network topology and the QN (134) information.
    - Resource (119): used to store and update point-to-point QKR information on each QKDL (132).
    - Services (117): used for QKD service creation, modification, deletion, storing and updating of the QKD service information.
    - Routing (118): used to compute and select the QKD path between QN pairs (134).
    - Strategy (116): used to perform QKR selection and assignment to satisfy the QKR requirements of QKD service requests.
    - QKMS (115): Management services for the Quantum Key generation, retention and distribution of QKR (202, 205, 207)

    Several QKMSs (115) can be constructed on the control plane (110) to manage the secret keys between QKD node (QN, given by 100) pairs, where the QKMSs are all controlled by the SDN controller and can manage secret-key exchange, storage, assignment, and destruction in a pair-wise manner.

    The SDN controller (111) with programmable and flexible network control capabilities can also provide the effective implementation technique for QKMSs. In general, an SDN controller is a centralized controller at a logical level, while QKD devices are distributed across the network and co-located with network nodes.

    The southbound (109) and northbound (103) interfaces cannot know the real secret keys, the security of the secret keys is assured.

3) Infrastructure Plane (130): The QNs (134) are interconnected by the QKDLs (132) in the infrastructure plane. OpenFlow agents are placed at each QN (134) to support an extended OFP and use a short-reach interface to communicate with the QN. The short-reach interface is inside the secure perimeter which enables the short-reach to communicate with the node's (i.e., the QN's) security perimeter. The short-reach interface is used to connect two devices at the same physical location. This location is securely contained and protected

within a security infrastructure. For example, a short-reach interface can be implemented using a cable for Internet Protocol (IP) connection between two co-located devices whose security is assured by protecting them within the physically secure security perimeter or location.

The communication between the SDN controller (111) and a QN (134) is achieved via protocol interpretation and messages in-between OpenFlow agents. This same communication between OpenFlow agents enables QNs to follow instructions from the SDN controller. Secret keys are constantly produced as per the secure QKD Rate (QKR) between two directly connected QNs.

The attainable QKR (202 in Fig. 2a) on each QKDL (132) is decided during the QKD network deployment.

A QN (134) and its respective OpenFlow agent together form the OpenFlow enabled QN, which is named QN-OF (134). Since all QNs referred to in this document are OpenFlow enabled, terms “QN” and “QN-OF” will be sued interchangeably in this work.

In contrast to prior art techniques, this architecture decouples QKD layer from the optical layer by constructing QKMSs (115) in the QKD layer. Two types of QKMSs, ones between the SDN controller (111) and each QN (134) and the ones between two nodes (QKD servers or QKDS given by 204 in Fig. 2a), are constructed to ensure the security of control messages via the control channels and services via the data channels, respectively. The QKD layer and the optical layer are controlled via southbound interface protocol (e.g., OpenFlow and NETCONF) given by 109, by the SDN controller (111) in control layer (110), where OpenFlow 1.4 protocol is used as southbound interface protocol (109).

The SDN controller with a programmable and flexible centralized network control manner can be used as the effective implementation technique for the control layer, which has been verified in recent research on SDN-controlled optical networks with time-shared QKD resources.

The QKD layer and optical layer share the optical fiber resources from physical WDM optical networks, where two wavelengths should be reserved as quantum channel (Q-Ch) and public channel (P-Ch) for constructing the OpenFlow-enabled QKMSs to jointly ensure the security of C-Chs and D-Chs. The remaining wavelengths can be used to support confidential service transmission.

OpenFlow-enabled optical cross connects (OXCs) are deployed in the optical layer in the Infrastructure plane (130).

The SDN controller is responsible for the management of entire network whereas the QKMS and OXCs are responsible for operating based on the instructions from the SDN controller. Service requests with different security requirements are triggered from the Application layer and interacts with the control layer via the RESTful API which is used as the northbound interface protocol.

Control channels and data channels may require different number of secret keys according to their security requirements. Specifically, this architecture can control and manage the network-wide secret-key resources in QKMSs to enable QKoD functionality in the QKDServ framework.

Note that the *secret keys will not be delivered* via the southbound and northbound interfaces. The secret keys are delivered from the secret-key servers of QKD nodes (QN) to the data communication equipment directly to satisfy the secret-key demands. The SDN controller and QKMSs on the control plane, as well as the QKD service users on the application plane are capable of obtaining the *real-time secret-key rate* and *volume* information, but they cannot know the real secret keys. The real secret keys are still stored in the secret-key servers (QKDS), given by 204, of

QKD nodes (QN), given by 134, which can be delivered to the QKD service creation process directly under the control of SDN controller (111). Thus, the security of secret keys is *guaranteed* in this architecture.

2

Fig 2a illustrates a QKDServ (Quantum Key Distribution Services) communication between Node-A and Node-B. The core principle of QKDServ is that an on-demand service can be offered to users over QKD network infrastructure (Fig. 1) where multiple users can apply for various QKD services to obtain their Quantum Key Services instead of having to deploy their own specific point-to-point QKD services for each and every connection. The extension of QKD technology from point-to-point to network-wide infrastructure necessitates enhancements in secret-key synchronization and options in storage and provisioning – this in turn should improve the performance of security and resource management.

In fig. 2a user A (101) requests a QKD service and associated QKR assignment (205). In order to achieve this, the QKD path is computed and selected by the algorithm in Fig 5.

In the next step, the required QKR of the QKD service is queried for each user, for example, in Fig. 2a, 3 kbps (205) and 4 kbps QKR (207) requests are generated for User A and User B respectively.

In case there are other QKD service requests, the available QKR (202) on the specific QKDL (132) is queried along the QKD path. The available QKR (202) is a real-time value based on the QKDL condition obtained by the QKD nodes reporting to the SDN controller (111) in real time.

If the available QKR can satisfy the QKR requirement of a QKD service request then the selection of the required QKR is performed from the corresponding QKDL for this service request; otherwise the specific QKD service request is rejected as per the algorithm shown in Fig. 5.

After the QKR generation is successfully accomplished, the QKD nodes use the produced secret keys on QKDL to generate the *session keys* to encrypt the session. The host of each site retrieves a *session key* from the locally hosted QKDS in the local QKD node (QN). Hosts obtain secure session keys and engage in encrypted individual communication sessions over classical (or conventional) network connection such as optical fiber, copper, wireless broadband etc.

Quantum secret key servers (QKDS) given by (204) are used to store the secret keys produced and relay it for end-to-end QKD service (Fig. 1).

The QKDServ example in Fig. 2a maintains a key repository, referred to as QKMS (Quantum Key Management Server) given by (115). The QKMS securely holds the key material generated via QKD.

The receiving device (User B, for example) or receiver (206) receives the encrypted information and uses the produced secret keys on QKDL to decrypt it. It can also share the produced secret keys with User A over the same QKDL. It is to be noted that the produced secret keys based on the individual QKR's should have the same secret-key size if one-time pad algorithm is used for ITS (Information Theoretic Security) encryption of secret keys. The transmitting device is given by 208.

Finally produced keys are assigned to User A or User B.

The generated secret keys between QKD Node-A and QKD Node-B can be stored in QKDS-A (204) and QKDS-B (204), which are embedded in Node-A and Node-B, respectively.

In Fig 2a, the generated secret keys are managed by QKMS to monitor the real-time remaining number of secret keys and provision secret keys between Node-A and Node-B.

Secret Key Allocation: In a QKD network, secret keys are constantly and uniquely produced without a re-use option. In this case, the concept of Quantum Key Rate (QKR) introduced here, is important since it is approximately 2 Mb/sec over 50 km standard optical fiber link as per technology available at the time of authoring this. The QKR (202, 205, 207) involves the generation of secret quantum keys in *bits per second* and is independent of any other network infrastructure resource. The secret keys are stored in the corresponding secure QKDS's which are part of each QN (134).

Management of the secret keys can be enhanced if a QKMS is virtualised between the participating Quantum Key nodes. The virtualised QKMS will then dynamically provide different number of secret keys for data encryption as per various corresponding security requirements.

The generation of secret quantum session keys via the QKMS (115) can be summarised as follows:

1. Quantum Node (QN) A (134) encodes raw keys in quantum signals and transmits them to Quantum Node B (134) via Q-Ch (quantum channel).
2. Quantum Nodes A and B exchange public information to accomplish secret-key synchronisation via the P-Ch (public channel).
3. The synchronised secret keys between Nodes A and B are stored in QKDS-A (204) and QKDS-B (204) respectively which are embedded in the respective nodes.
4. The secret keys between QKDS-A and QKDS-B are virtualised to construct QKMS which enables the distribution of QKD keys on demand according to different security demands between Nodes A and B through the control channel or data channel.

The QKMS construction process is summarized in three main steps:

(1) Quantum communication node (QN)-A encodes raw keys in quantum signals and transmits them to QN-B via the quantum channel (Q-Ch). QN is an aggregating device that contains quantum transceiver/receiver combination.
(2) QN-A and QN-B interchange public information to accomplish secret-key synchronization via the public channel (P-Ch).
(3) The synchronized secret keys between QN-A and QN-B are stored in QKDS-A and QKDS-B which are embedded in Node-A and Node-B, respectively. The secret keys between QKDS-A and QKDS-B are virtualized to construct the QKDL (132), which enables QKoD according to different security requirements between Node-A and Node-B through the control channel (C-Ch) or data channel (D-Ch).

Besides QKD enhancements, also important are wavelength allocation for QKMSs and optically supported software-defined networks are important for QKMS construction: Two wavelength channels should be reserved as Q-Ch and P-Ch for secret-key synchronization in WDM support in the optical underlay for the SDN. The control messages are transmitted by IP-based routing through the control channels in software-defined networks (SDN), and hence, control channels will occupy IP channels rather than data channels. Q-Ch, P-Ch and D-Ch can be placed at C-band (1530−1565 nm) to achieve lower attenuation performance and high compatibility with the existing WDM optical networks.

Additionally, physical layer impairments must be considered while allocating wavelengths for Q-Ch and P-Ch. Raman scattering can be reduced by placing the Q-Ch at the highest frequency. Moreover, four-wave-mixing (FWM) effect can be minimized by reserving 200 GHz spaced bandwidth between Q-Ch and classical channels (i.e., P-Ch and D-Ch).

Prior art techniques for QKD networks mainly focussed on deployment of point-to-point links, where a pair of QNs is required for each connection. Recently, *time-scheduled technique* has been proposed to reduce the financial expenditure of deploying a QKD network . Hence, based on the specific requirement of QKMS construction over a software-defined optical network, a single QN (134) can be enabled to be time-shared between multiple endpoints allowing the establishment of multiple QKD connections using fewer QNs. This technique reduces the cost of QKMS (115) construction over an SDN network because of the significant reduction in hardware and the efficient usage of QNs.

Secret-key resources in QKMSs are finite and precious. Hence, the secret-key assignment problem for the C-Chs and D-Chs should be addressed while implementing QKoD as a function of QKDServ. In SDN, the control messages via the control channels are usually transferred at megabit-per-second data rate, which are lower than the data complexity of attacks. Hence, to enhance the security performance of C-Chs, key-assignment and key-updating are performed for control messages between the SDN controller and each node through the QKDL of a service. Each node along the QKDL is configured by the SDN controller, and then, the control messages are sent to each node via the control channels. Based on the specific security requirement of control channels, QKMS for C-Chs can allocate the required number of secret keys for SDN controller and nodes to secure the control messages via the control channels.

Different number of secret keys (denoted as $Key_{x\text{-}y}$, where *x* is the node number and *y* is the service number) can be assigned for control channels between the SDN controller and different nodes along different QKD links (QKDL).

In an embodiment in Fig. 2b, $Key_{1\text{-}1}$ (and $Key_{2\text{-}1}$) are assigned for control channels between the SDN controller and Node 1 (and Node 2) through the lightpath Node 1→2 of Service 1; $Key_{1\text{-}2}$ (and $Key_{2\text{-}2}$, $Key_{3\text{-}2}$) are assigned for control channels between the SDN controller and Node 1 (and Node 2, Node 3) through the QKDL Node 1→3 of Service 2.

For the services via the data channels, the required number of secret keys is related to the key-length and key-updating period. Based on the specific security requirement of data channels, QKMSs for D-Chs can allocate the required number of secret keys for the source and destination nodes to secure the services via the data channels. For example, for three services (denoted as r_1, r_2, and r_3) with different security requirements (including keylength and key-updating period) the time complexity of attacks for key updating (i.e., the maximum available time for a secret key, denoted as T_y) and data complexity of attacks for key updating (i.e., the maximum data size that can be encrypted by a secret key, denoted as D_y) give the required key lengths of r_1, r_2, and r_3 as 128 bit, 192 bit and 256 bit, respectively for the required key-updating periods of r_1, r_2, and r_3 are T_1, T_2, and T_3 ($T_1 < T_2 < T_3$) respectively , or D_1, D_2, and D_3 ($D_1 < D_2 < D_3$) respectively. Hence, the service with longer key-length and shorter key-updating period shows higher security level and requires more secret keys to be assigned for data encryption.

Both in cases of Fig. 2a and 2b, once the QKD authentication process is complete, the users communicate using a BB84 type (Fig. 8 prior art) Quantum Communication protocol.

3

Protocol Extension of OpenFlow 1.4 (Figs. 3a-f): This architecture leverages quantum metadata communication (301, 302, 303, 304, 305, 306) in the SDN managed by utilizing its compatibility with the OpenFlow 1.4 protocol. Controllers today leverage OpenFlow functionalities. Because of its flexibility in programmability and compatibility with management of optical networks, OpenFlow is highly suitable to control the new attributes defining the classical channel that carries metadata between various quantum aware stacks/devices. In particular, OpenFlow version 1.4 extensively supports attributes specific to lambda-switching in optical networks that allows the ability to manage quantum optical channels interconnecting quantum network devices.

Fig 3a-3f show the OpenFlow 1.4 extension needed to enable QKD service creation (104), modification (106) and deletion (108), since OFP, in its original format, can only support packet-switching in the electrical domain. New messages are attached to the OFP 1.4.0 header for the ease of implementation in an OpenFlow based platform via the use of YANG tools.

Fig 3(a)-3(f) shows the main structure of extended OFP messages for QKD service creation request (405 in Fig. 4) and response, QKD service modification request and response, and QKD service deletion request and response.

The OFP v1.4.0 header is composed of version, type, length, and xid. The version field indicates the OFP version being used. The type field indicates the type of the message. In this architecture, the message types of QKD service creation, modification and deletion requests and responses are configured as 32 (301, 302, 303) and 33 (304, 305, 306), respectively. The length field indicates the total length of the message. The xid which depicts the transaction ID, is a unique value used to match requests to responses.

In addition to the OFP v1.4.0 header, 32 bits were added for QKD service ID (311) to represent a QKD service and 32 bits for QN-OF ID (321). Additionally, another 32 bits were added for message function to exhibit the different functions of extended OFP messages. For example, 0x0000FFFF (331), 0xFFFFFFFF (341), and 0xFFFF0000 (351) to indicate the message functions of QKD service creation, modification, and deletion requests and responses, respectively. Another 32 bits were for both the input port (371) and output port (381) of each QNs (134) along the QKD path of this QKD service.

A set of 128 bits were added for QKR slot ID (361) to describe the usage of 128 QKR slots (e.g., 1 is busy and 0 is free). Based on the number of attainable QKR slots on each QKDL, the number of bits for QKR slot ID can be changed. As shown in Figs. 3(a) and 3(c), the QKR slot usage after QKD service creation and modification can be different, since a user can request to modify its QKR requirement. Also, as illustrated in Figs. 3(b), 3(d), and 3(f), another 32 bits were added for QKD service responses to indicate the status of QKD service creation (104), modification (106) and deletion (108) process.

4

Communication Workflow: Fig. 4 shows the communication workflow among the three architectural planes of the QKDServ where the intercommunication for successful QKD service creation is depicted.

As a first step, a service request (405) comes in from user(s) (101) in the application layer (100) and the SDN controller (111) in the control plane configures all the QN-OFs on the infrastructure plane to accomplish QKD network initialization.

After establishment of TCP session, the SDN controller achieves OpenFlow handshake and exchanges keepalives with all the QN-OFs (134). The SDN controller initializes all the QN-OFs and acquires detailed information of each QN-OF. Based on the instructions from QKD network operator on the application plane, the SDN controller configures (406, 407, 408, 409, 410, 411) point-to-point QKD connections via connection of QN-OFs (134) to the corresponding QN-OFs using QKDLs (132) to constitute the QKD network.

After the QKD network is up and running, the QKD network topology and QKR on each QKDL is automatically reported to the SDN controller (412). After QKD network initialization, QKD network operator generates multiple QKD service requests according to the QKR requirements of different users, and sends these requests to the SDN controller. The QKDServ involves the creation (104), modification (106), and deletion (108) of QKD services.

Upon receiving a QKD service creation request (405), the SDN controller computes and selects the QKD path between the *source* QN-OF and *destination* QN-OF. Then, SDN controller searches the real-time available QKR slots on each QKDL along the selected QKD path and selects the required QKR slots.

If the real-time available QKR slots are capable of satisfying the QKR requirement of this request, the SDN controller configures the source OF-QUN, intermediate QN-OF(s), and destination QN-OF along the QKD path to accomplish SKR slot assignment for QKD service creation. Otherwise this QKD service creation request will be rejected.

The intermediate QN-OF(s) along the selected QKD path can relay the secret keys for long-distance end-to-end QKD. After accomplishing QKR slot assignment for QKD service creation, a success response will be sent back to the QKD network operator.

In addition, when the QKR requirement of a user changes, the established QKD service for this user needs to modify its QKR requirement.

Upon receiving a QKD service modification request, SDN controller first performs the steps 3 and 4 numbered in Fig. 4. If the real-time available QKR slots along the QKD path can satisfy the modified QKR requirement, QKR slot re-assignment (i.e., steps 5 to 10 numbered in Fig. 4) will be performed for QKD service modification. Otherwise the QKD service modification request will be rejected.

In addition, optionally, QKD network management (102) operator can request the deletion of a QKD service when the QKD service has expired (i.e., at the end time of QKD service request). Upon receiving a QKD service deletion request, SDN controller configures the QN-OF(s) to stop assigning QKR slots to this QKD service and deletes the information of this QKD service.

5

Routing and QKR Assignment Strategy for QKD Service Creation: Fig 5 presents the routing and QKR assignment strategy for QKD service creation and modification. The overall objective of this strategy is to accommodate the QKD service requests successfully for multiple users as many as possible over a QKD network.

The QKD network model and QKD service request model is as follows:

The QKD network topology is denoted by $G(V_Q, V_T, K)$, where V_Q, V_T and K denote the sets of QN-OFs, QKDLs, and attainable QKR slots on each QKDL, respectively.

Assumption is that the number of attainable QKR slots on each QKDL is the same, which is fixed to $|K|$. The set of incoming QKD service requests is denoted by R.

A QKD service request is modelled (501) as $r(s_r, d_r, t_s, t_e, k_c, m, k_m)$ where s_r, d_r, t_s, and t_e denote the *source* QN-OF, *destination* QN-OF, *start time*, and *end time* of QKD service request r, respectively.

The required number of QKR slots for QKD service (corresponding to r) creation is denoted by k_c. The number of times for the modification of QKD service (corresponding to r) within the holding time is denoted by m.

The changed number of QKR slots for modifying QKD service (corresponding to r) each time is denoted by k_m. Note that the value of k_m can be positive or negative, since the required number of QKR slots for modifying QKD service each time can be increased or decreased.

At first, Dijkstra algorithm is utilized (502) to compute and select the shortest QKD path between the source QN-OF and destination QN-OF of QKD service request, since the shortest QKD path is beneficial to reduce the required number of QKDLs and QKR slots for QKD service creation. The worse-case complexity of routing is $O(|V_Q \cup V_T|^2)$.

Then, the real-time available QKR slots on each QKDL along the selected QKD path are computed, which can be obtained by the QN-OF reporting to the SDN controller in real time (503).

If the QKR requirement of this QKD service creation can be satisfied (504), the first fit (FF) algorithm is utilized (505) to select and assign the required SKR slots for QKD service creation, otherwise this QKD service request is rejected (506).

The resource module in the SDN controller is used to store and update point-to-point QKR information (including the attainable and real-time available QKRs) on each QKDL (507, 513).

The FF algorithm is chosen in this case due to its low complexity and small computation overhead. Using FF algorithm, all the real-time available QKR slots are numbered (508), in which a QKR slot with small serial number is selected and assigned before a QKR slot with large serial number (509).

In this case, all the QKD service requests are treated with equal weights and process the QKD service requests one by one with time. This is reasonable since we assume that QKD service requests arrive dynamically queuing delay is not an option for any of those requests. Also, if a concurrency situation occurs, the QKD service requests will be prioritised randomly and processed accordingly.

QKR reassignment (510) is performed for QKD service modification (514).

If the QKR requirement of this QKD service modification is satisfied within the holding time, FF algorithm is performed again (515) to select and assign the required SKR slots for QKD service modification (106), otherwise this QKD service request is rejected (512).

The last step (515) ends the QKD service request process.

Routing and Key Assignment: Efficient implementation of QKD on Demand (QKoD) in the QKDServ architecture utilizing SDN requires a novel routing and key assignment/distribution strategy.

In this work, a QKD Routing, Key and Wavelength Assignment (QRKWa) algorithm is proposed. QKMS-D-Ch for D-Ch's are built as needed between any pair of nodes and QKMS-C-Ch's for C-Ch's are built between any node and the SDN controller; hence, the number of QKMS-D-Ch and QKMS-C-Ch are $n(n-1)/2$ and n respectively.

Different services over the data channels will need different key lengths and key-updating time periods. Hence, Eq. (1) and (2) below gives the security requirement matrices based on key-updating period dependent on the time complexity and data complexity of attacks respectively.

$$SL_T = \begin{bmatrix} u_{l_1,T_1} & u_{l_1,T_2} & \dots & u_{l_i,T_k} \\ u_{l_2,T_1} & u_{l_2,T_2} & \dots & u_{l_i,T_k} \\ u_{l_3,T_1} & u_{l_3,T_2} & \dots & u_{l_i,T_k} \end{bmatrix} \quad (1)$$

$$SL_D = \begin{bmatrix} u_{l_1,D_1} & u_{l_1,D_2} & \dots & u_{l_i,D_k} \\ u_{l_2,D_1} & u_{l_2,D_2} & \dots & u_{l_i,D_k} \\ u_{l_3,D_1} & u_{l_3,D_2} & \dots & u_{l_i,D_k} \end{bmatrix} \quad (2)$$

Key-assignment and key-updating are performed for control messages between the SDN controller and each node through the path of the service. Hence, the total number of secret keys required to ensure the security of control messages via the control channels for service request r is given by Eq. (3) below

$$N_{rc} = l_i\,(h_r + 1) \quad (3)$$

The number of secret keys for key-updating, taking into account time and data complexities of attacks, are given by Eq. (4) and (5) as follows:

$$N_{rt} = \frac{l_i\, t_h}{T_k} \quad (4)$$

$$N_{rd} = \frac{l_i\, t_h b_r}{D_k} \quad (5)$$

Table 1 in Fig. 6a shows the proposed QRKWa algorithm. Table 2 in Fig 6b lists the definitions for the symbols in Table 1.

Table 1 depicts a secure key on demand and an algorithm to govern routing, wavelength allocation and the key assignment model. The notations and definitions of corresponding terminology and parameters are shown in Table 2.

The QRKWa algorithm is divided into three steps, i.e., QKD routing and wavelength assignment (QRWa) for services via the data channels, key assignment (Ka) for control messages via the control channels and key assignment for services via the data channels.

Note that the control channels occupy IP channels, and hence, QRWa step is not needed for control channels. Two cases, specifically, *time complexity* of attacks and *data complexity* of attacks are considered for *key-updating* in *key assignment* for data channels.

In *step 1*, routing computation is done with *K-shortest-path* Dijkstra's algorithm, and the wavelengths reserved as data channels are allocated to the service with First Fit (FF) algorithm. The service can establish a QKDL path with the assigned wavelength channel. The security requirement of a service is represented by the required key-length and key-updating period as defined in Eqs. (1)–(2), and the total required number of secret keys for the control channels and data channel of each service is calculated by Eqs. (3)–(5).

In order to satisfy the security requirement, key assignment for control channels and data channel of each service is performed in *step 2* and *step 3*, respectively. In *step 2* and *step 3*, secret keys stored in QKMSs are allocated for the corresponding control channels and data channels by utilizing FF algorithm, respectively.

Note that the secret keys in the QKMSs (115) are stored in the unit of bit and cannot be reutilized, and hence, the FF algorithm with simple implementation advantage can provide relatively good performance in key assignment.

Lines 2 to 38 accomplish the QRWa for control channels and data channel of service request r. Considering the worst condition, the time complexity of step 1 (lines 2 to 14), step 2 (lines 15 to 22) and step 3 (lines 23 to 38) are $K|V|^2 + K|W|$, $|Q_c|$ and 1, respectively. Hence, the total time complexity of QRKWa algorithm for service request r is approximately $O(K|V|^2)$.

1. The entry point for manufacturers is the fact that they will only need to produce the device/solution without the need for the whole network and all its associated device to be "quantum aware". The controller will manage the specific requirements, interactions and configure their capabilities.
2. The specific data that the QKD environment will produce, i.e., the keys, can be managed by the controller itself. This means that if it is a "forwarding key" or a control plane security key then the controller can manage its associations and hand it over when and as appropriate to another entity such as the QKMS via API transparently.
3. Telecommunications industry perspective is the controller will run inside a trusted domain which is a secure environment from a physical and logical point of view – hence using trusted repeaters does not involve an extra security assumption. Different QoS can still be distinguished and managed accordingly.
4. For a start, this solution can be implemented in a small-scale quantum network and then expanded in an incremental manner.
5. The quantum part of the larger network can be upgraded as and when technologies become available. A hybrid of classical and quantum switching network has been tested. This can be used for starter solution and expanded as and when new technology becomes available.
6. A hybrid usage of quantum and classical crypto can be easily implemented, with AES 256 type of algorithm for the classical part.
7. From the perspective of a network operator, the integration of SDN type environment could reduce CPEX because the deployment of a QKD pair for each point-to-point will not be

necessary provided a transmitter can be time-shared among various receivers – hence number of QKD devices could be reduced.

8. And lastly, the network itself can be a user of the QKD keys.

Whereas QKD can provide end-to-end encryption based security which is immensely improved from what we have today, Q-Ch's and P-Ch's should be protected simultaneously in a QKD over SDN environment. Due to the key updating period at different time slots for different services, security level protection will occur at sub-wavelength level.

Budget Estimate demands that two types of nodes should be deployed in a QKD environment: QKD node and intermediate node with trusted repeaters following ITS convention. In practice variation in number of nodes and types of channels will require different budget estimates and associated efficiency of the QKD implementation.

Consideration of Key Generation is needed in regards the following: The generation rate of secret keys in bits per second in current QKD systems is low compared with the gigabit per second data transmission. Whereas this is not a bottleneck for networks within 50 km of distance, this will add to power consumption if and when the intermediate nodes are implemented in order to increase distance efficiency as increasing the number of nodes and Q-Ch's/P-Ch's/C-Ch's will further increase the required key rate.

In regards to tools, Google released Cirq as opensource in July 2018. It is envisaged that either Cirq or TensorFlow or both can be leveraged on GPU driven hardware (NVIDIA for example) and/or on HyperFlex AI starter kit for the coding side while either Google's D-Wave based computing infrastructure can cover the quantum chipset requirement. Another option is IBM Q-Experience lab (who has made their superconductor based 16 qubit computing available for all research) can be leveraged for the actual backend quantum computing. Both IBM and Google have publicly expressed their strong intention to move forward towards Universal Quantum Computing, the next goal being 50 qubit computation capability. A third option is Artiste and a still fourth option is Rigetti who has created a Python library called "Pyquil".

7 DRAWINGS

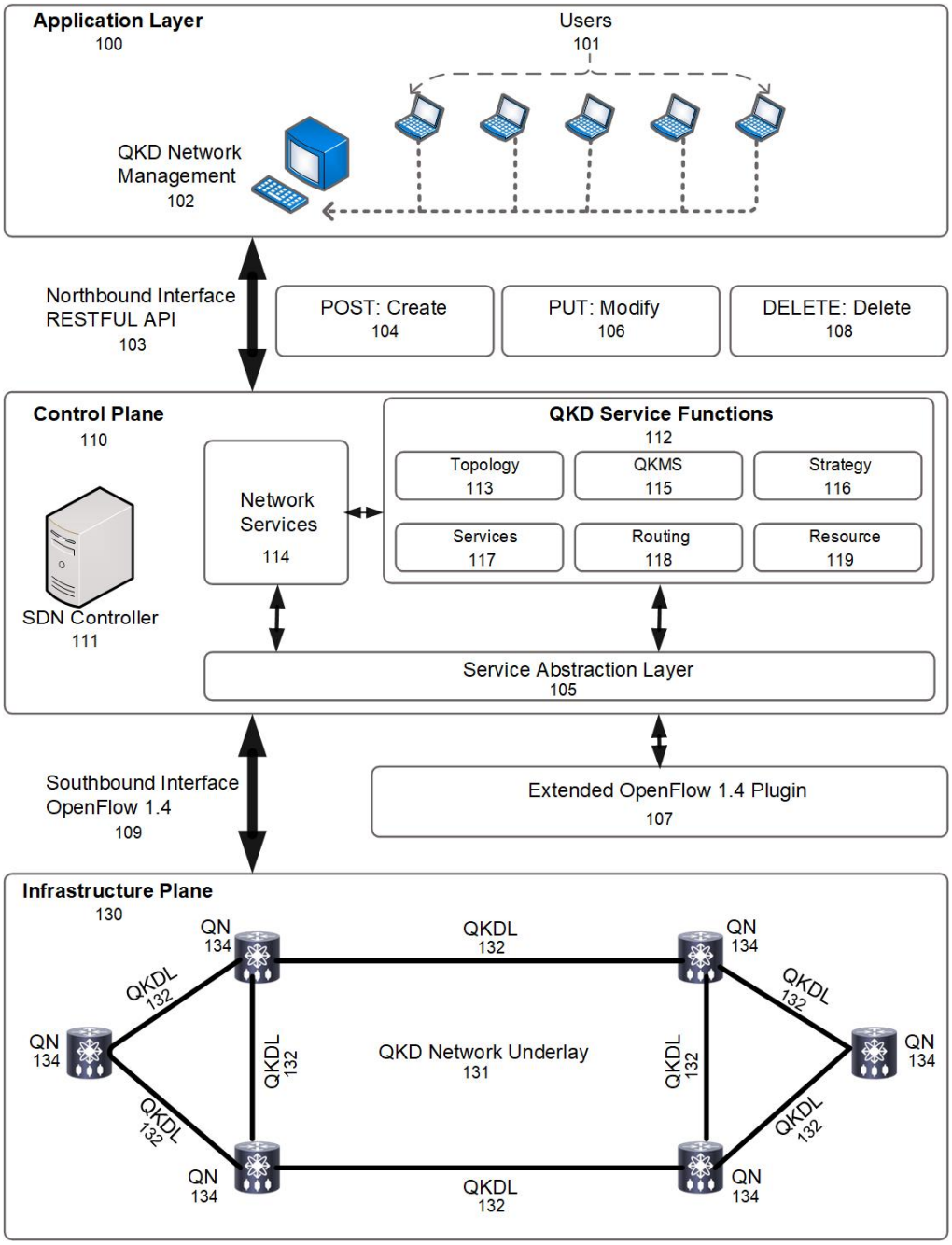

Application Layer
100
Users
101
QKD Network
Management
102
Northbound Interface
RESTFUL API
103
POST: Create
104
PUT: Modify
106
DELETE: Delete
108
Control Plane
110
QKD Service Functions
112
Topology
113
QKMS
115
Strategy
116
Network
Services
114
Services
117
Routing
118
Resource
119
SDN Controller
111
Service Abstraction Layer
105
Southbound Interface
OpenFlow 1.4
109
Extended OpenFlow 1.4 Plugin
107
Infrastructure Plane
130
QN
134
QKDL
132
QKD Network Underlay
131

Fig 1

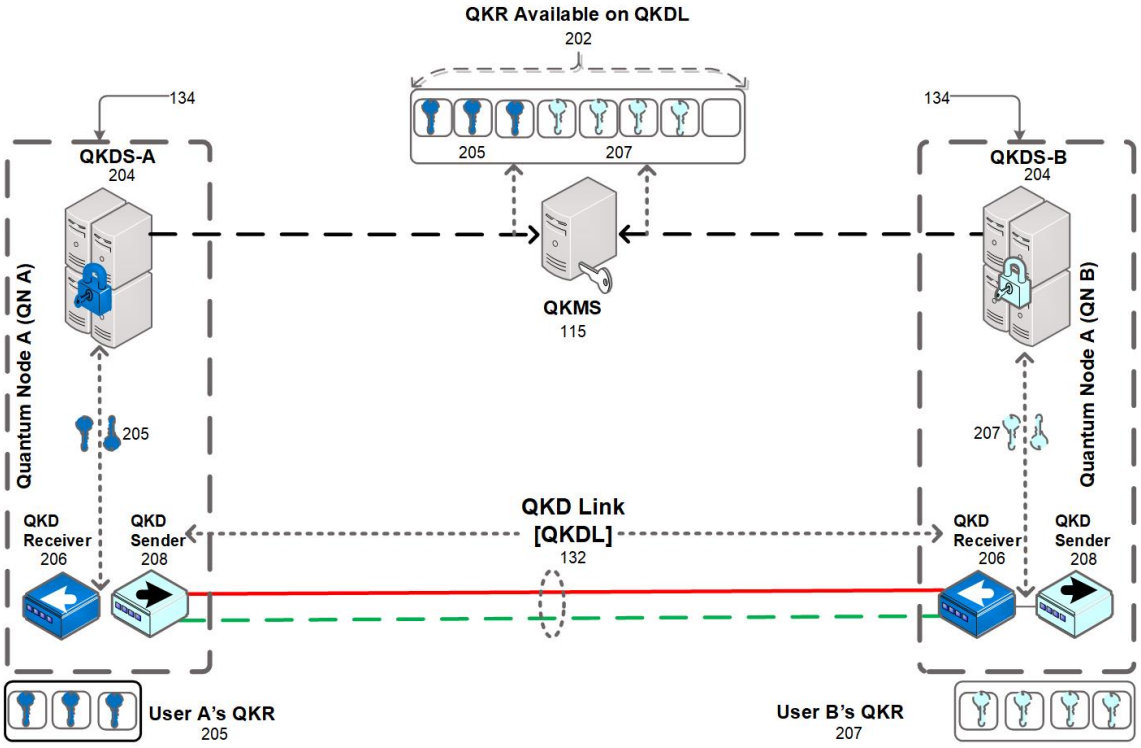

QKR Available on QKDL
202
205
207
134
QKDS-A
204
QKMS
115
QKDS-B
204
Quantum Node A (QN A)
205
207
Quantum Node A (QN B)
QKD Receiver
206
QKD Sender
208
QKD Link
[QKDL]
132
QKD Receiver
206
QKD Sender
208
User A's QKR
205
User B's QKR
207

Fig 2a

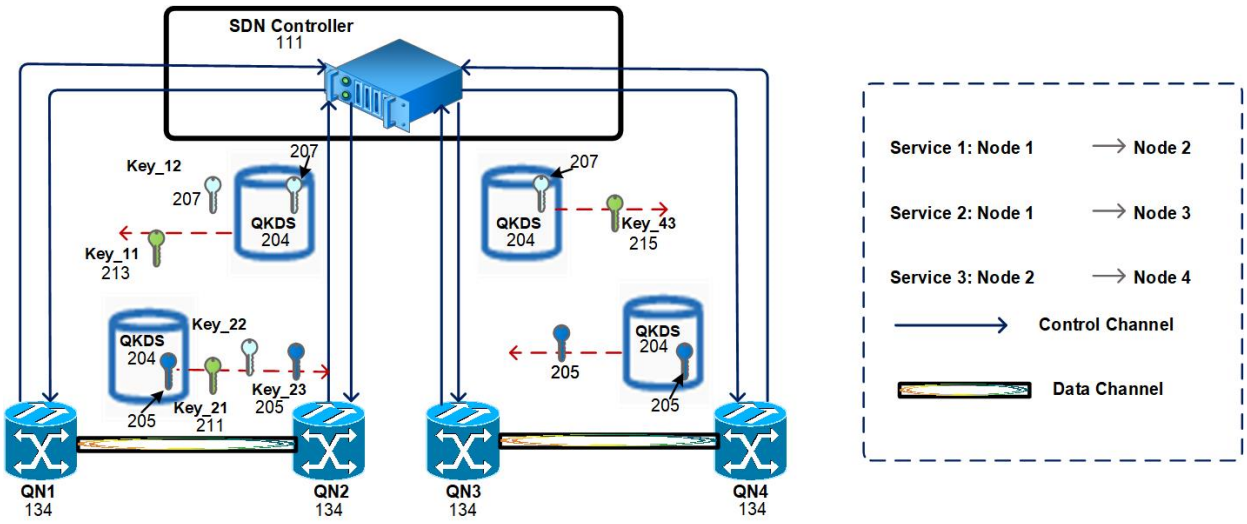
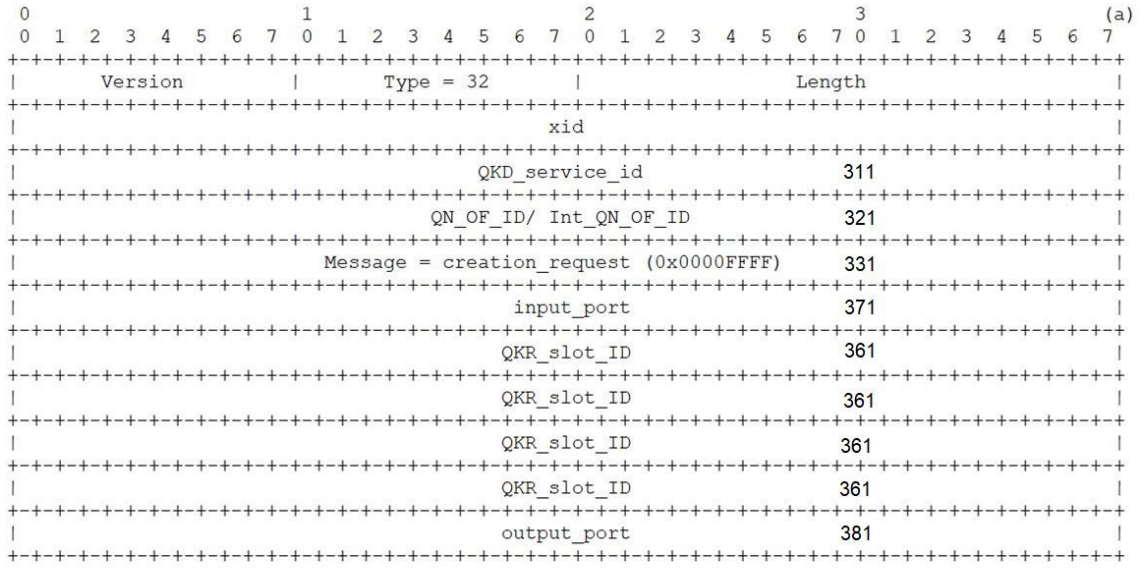
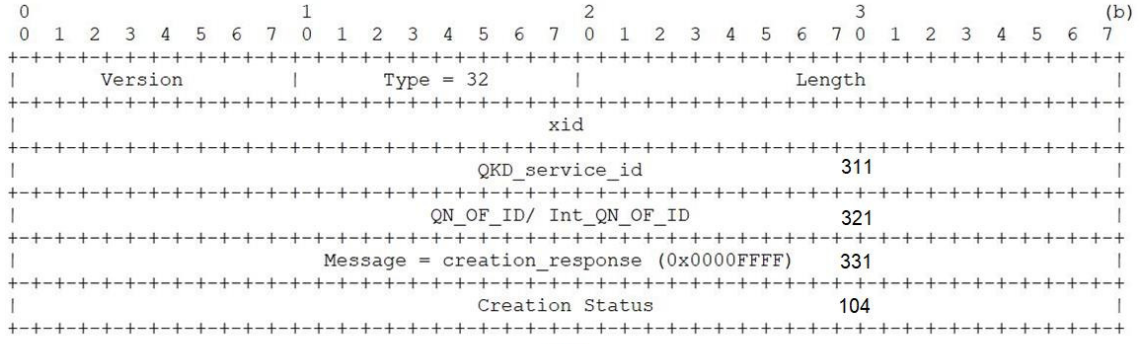

SDN Controller
111
Key_12
207
207
QKDS
204
Key_11
213
207
QKDS
204
Key_43
215
QKDS
204
Key_22
Key_23
205
Key_21
211
205
205
QKDS
204
205
205
QN1
134
QN2
134
QN3
134
QN4
134
Service 1: Node 1
Node 2
Service 2: Node 1
Node 3
Service 3: Node 2
Node 4
Control Channel
Data Channel

Fig 2b



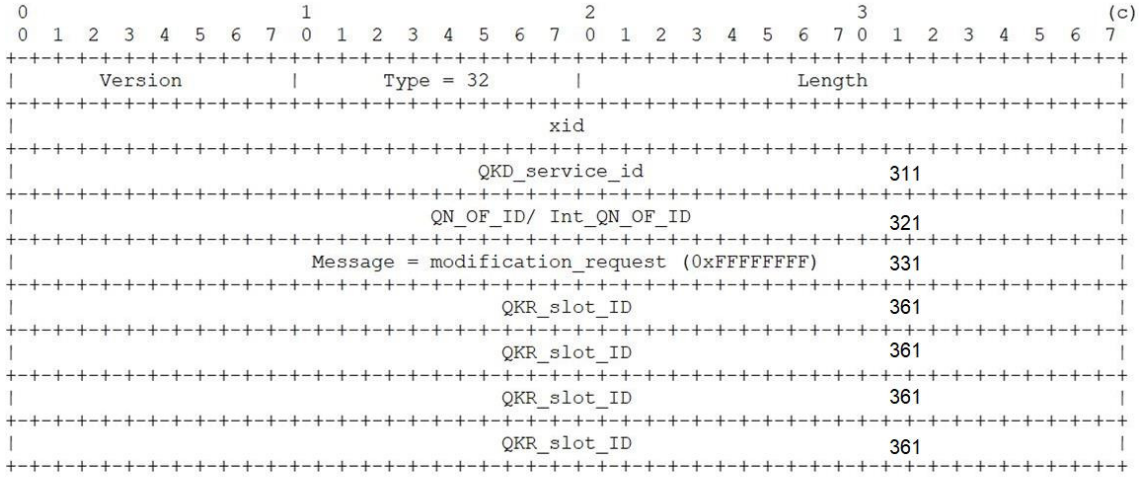
301

Fig 3a



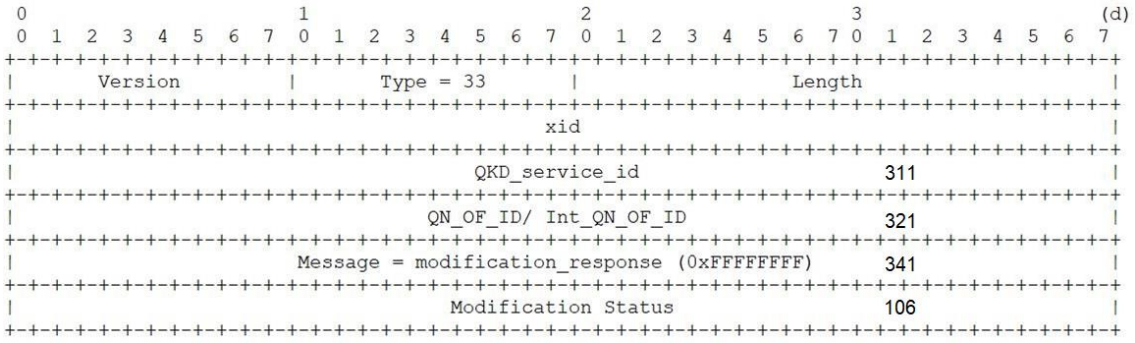
302

Fig 3b



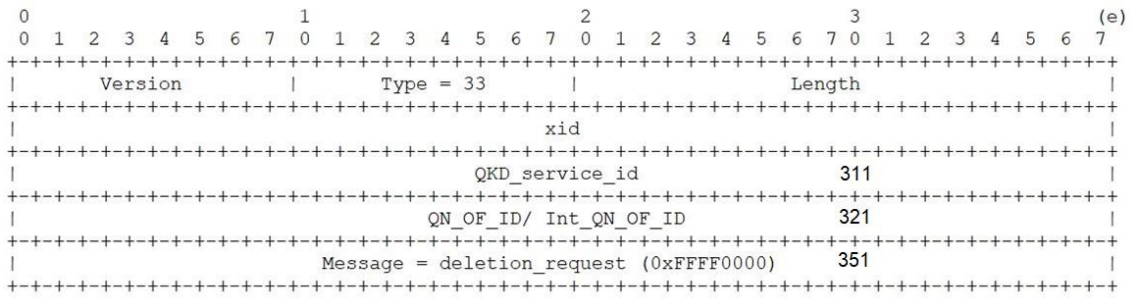
303

Fig 3c



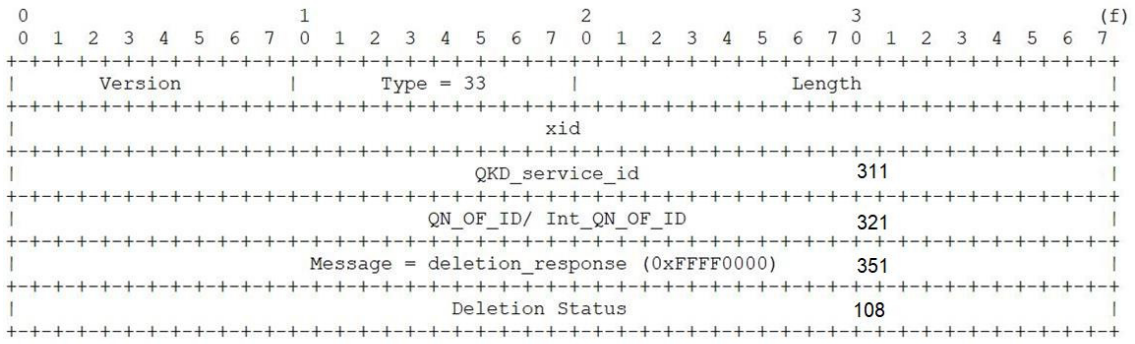
304

Fig 3d



305

Fig 3e



306

Fig 3f

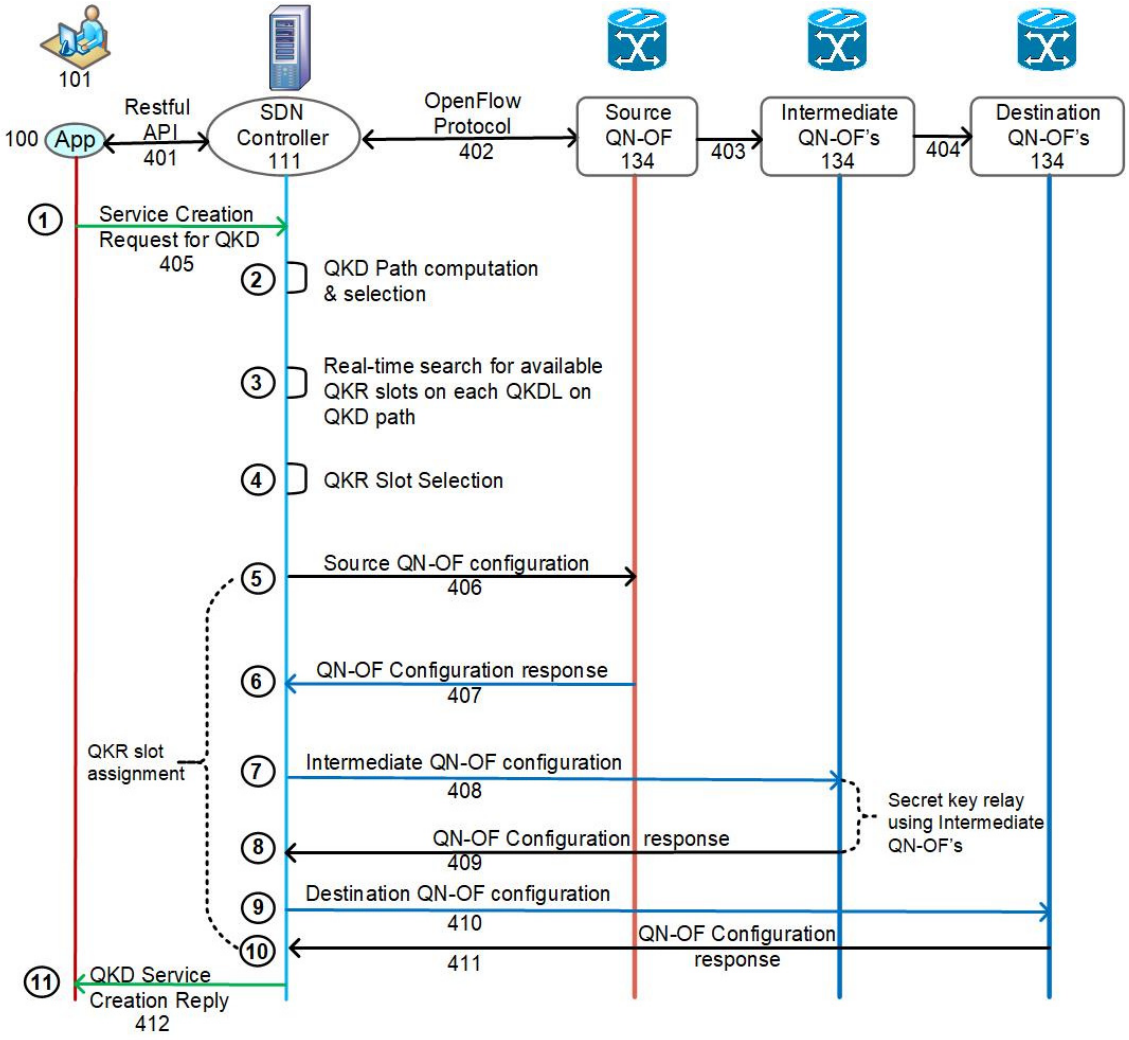

101
100
App
Restful API 401
SDN Controller 111
OpenFlow Protocol 402
Source QN-OF 134
403
Intermediate QN-OF's 134
404
Destination QN-OF's 134
1
Service Creation Request for QKD 405
2
QKD Path computation & selection
3
Real-time search for available QKR slots on each QKDL on QKD path
4
QKR Slot Selection
5
Source QN-OF configuration 406
6
QN-OF Configuration response 407
QKR slot assignment
7
Intermediate QN-OF configuration 408
Secret key relay using Intermediate QN-OF's
8
QN-OF Configuration response 409
9
Destination QN-OF configuration 410
10
QN-OF Configuration response 411
11
QKD Service Creation Reply 412

Fig 4

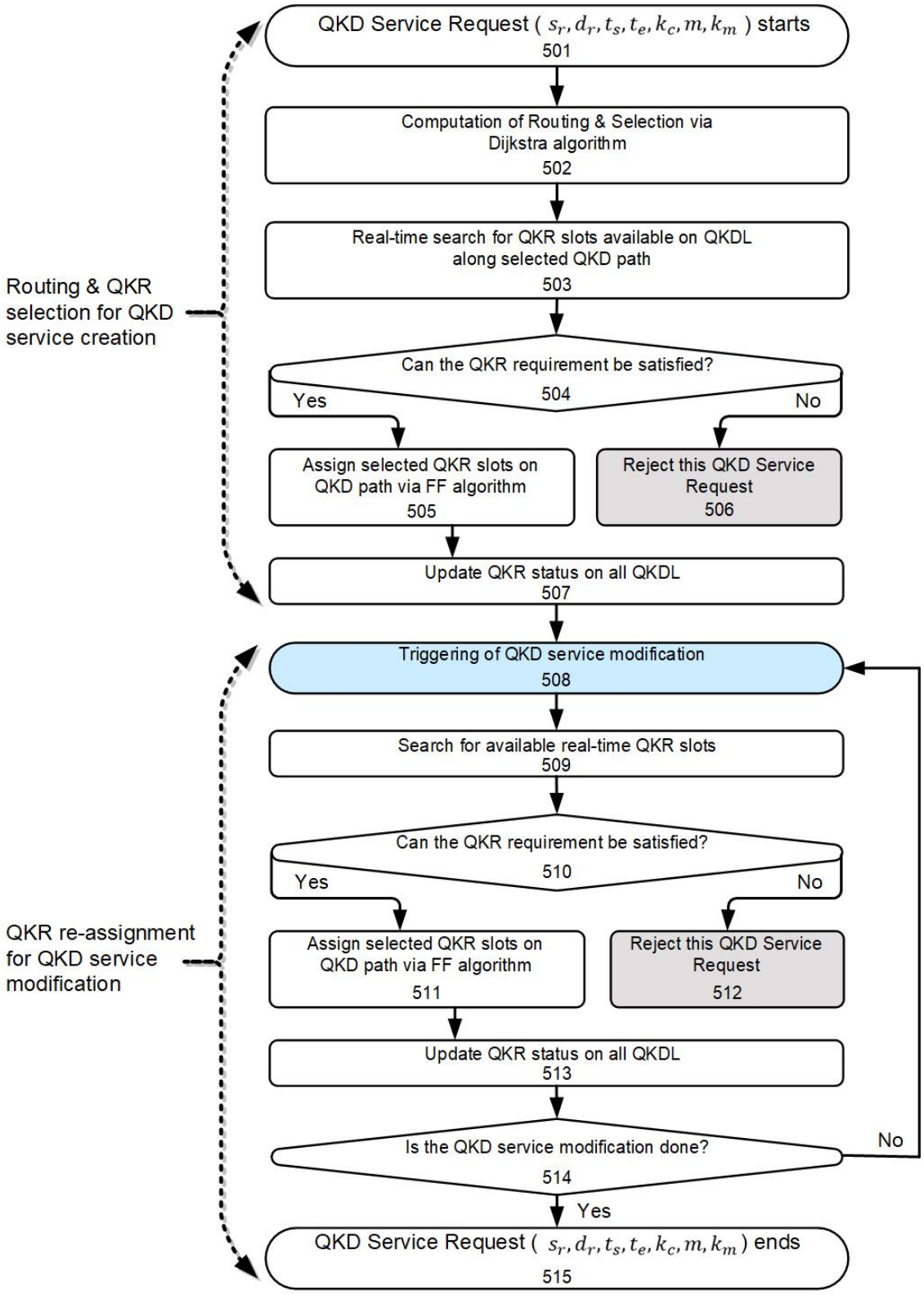

QKD Service Request ($s_r, d_r, t_s, t_e, k_c, m, k_m$) starts
501
Computation of Routing & Selection via Dijkstra algorithm
502
Real-time search for QKR slots available on QKDL along selected QKD path
503
Routing & QKR selection for QKD service creation
Can the QKR requirement be satisfied?
504
Yes
No
Assign selected QKR slots on QKD path via FF algorithm
505
Reject this QKD Service Request
506
Update QKR status on all QKDL
507
Triggering of QKD service modification
508
Search for available real-time QKR slots
509
Can the QKR requirement be satisfied?
510
Yes
No
QKR re-assignment for QKD service modification
Assign selected QKR slots on QKD path via FF algorithm
511
Reject this QKD Service Request
512
Update QKR status on all QKDL
513
Is the QKD service modification done?
514
No
Yes
QKD Service Request ($s_r, d_r, t_s, t_e, k_c, m, k_m$) ends
515

Fig 5

Table 1. QKD Routing, Key and Wavelength Assignment (QRKWa) algorithm

Algorithm: QKD Routing, Key and Wavelength Assignment (QRKWa) algorithm		
Input: $G(V, E, W, Q_c, Q_d), R, N_c, N_d, SL_T, SL_D$ **Output**: QRWa (QKD Routing Wavelength Assignment) for each service, QKA (Quantum Key Assignment) for C-Ch's of each service, QKA for D-Ch's of each service l_i		
Start	1	**for** each service request $r(s_r, d_r, b_r, t_h, u_{l_i,T_k}, u_{l_i,D_k})$, **do**
	2	routing computation with K-shortest-path algorithm;
	3	**for** each computed alternative path P_r, **do**
	4	search available wavelength resources through the path:- $W(P_r) = \{W_1, W_2, W_3, ..\}$;
	5	**if** $W(P_r) \neq \emptyset$, **then**
QRWa for each service	6	select an available wavelength with *First Fit* algorithm;
	7	update link wavelength resource status;
	8	**break;**
	9	**else**
	10	**continue;**
	11	**end for**
	12	**if** all the computed *K* alternative paths have no available wavelength resource, **then**
	13	the service request is blocked;
	14	**end if**
	15	**for** each node v_j through the selected path
	16	**if** $l_i \leq N_c^{v_j}$ (security requirement can be satisfied), **then**
	17	select l_i (bit) secret keys from the QKMS-OF-C-Ch for node v_j with *First Fit* algorithm;
QKa for C-Ch's of each service	18	update the remaining numbers of secret keys $N_c^{v_j}$ in the QKMS-OF-C-Ch for v_j node;
	19	**else**
	20	security requirements cannot be satisfied and the C-Ch's of this service is insecure;
	21	**break;**
	22	**end for**
	23	**if** $u_{l_i,T_k} \in SL_T$ and u_{l_i,D_k} = NULL, **then**
	24	compute the number of required secret keys N_{rt} for the D-Ch of this service;
	25	**if** $N_{rt} < N_{sd}$ (security requirement can be satisfied), **then**
QKa for D-Ch's of each service [Case 1]	26	select N_{rt} (bit) secret keys from the QKMS-OF-D-Ch between node s_r and node d_r with *First Fit* algorithm;
	27	update remaining secret keys N_{rd} in QKMS-OF-D-Ch between node s_r and node d_r;
	28	**else**
	29	security requirements cannot be satisfied and the D-Ch of this service is insecure;
	30	**end if**
	31	**if** $u_{l_i,D_k} \in SL_D$ and u_{l_i,T_k} = NULL, **then**
	32	compute the number of required secret keys N_{rd} for the D-Ch of this service;
QKa for QKa for D-Ch's of each service [Case 2]	33	**if** $N_{rd} < N_{sd}$ (security requirement can be satisfied), **then**
	34	select N_{rd} (bit) secret keys from the QKMS-OF-D-Ch between node s_r and node d_r with *First Fit* algorithm;
	35	update remaining secret keys N_{sd} in QKMS-OF-D-Ch between node s_r and node d_r;
	36	**else**
	37	security requirements cannot be satisfied and the D-Ch of this service is insecure;
	38	**end if**
End	39	**end for**

Fig 6a

Table 2

Notations and Definitions
$G(V, E, W, Q_c, Q_d)$: QKD over SDN topology
$V = \{v_1, v_2, v_3, \dots, v_n\}$: ser OpenFlow OXC node
$E = \{e_1, e_2, e_3, \dots, e_n\}$: set of fiber link
$W = \{w_1, w_2, w_3, \dots, w_n\}$: set of wavelength resource on each link reserved for data channel
$Q_c = \{Q_1, Q_2, \dots Q_n\}$: set of QKMS-OF-C-Ch
$Q_d = \{q_1, q_2, \dots q_{n(n-1)/2}\}$: set of QKMS-OF-D-Ch
N_c: initial number of secret keys stored in each of QKMS-OF-C-Ch for node v_j
N_d: initial number of secret keys stored in each QKMS-OF-D-Ch between nodes s and d
$N_c^{v_j}$: remaining real-time secret keys stored in each of QKMS-OF-C-Ch for node v_j
N_{sd}: remaining real-time secret keys stored in each of QKMS-OF-D-Ch between nodes s and d
R: Service request sets
$r(s_r, d_r, b_r, t_h, u_{l_i,T_k}, u_{l_i,D_k})$: The service-request.
s_r: source node of service request r
d_r: destination node of service request r
b_r: bandwidth demand associated with service request r
t_h: holding time of service request r
h_r: number of hops of service request r
u_{l_i,T_k}, u_{l_i,D_k}: security requirement of service request r
l_i : options for key lengths from $\{l_1 = 128\ bit;\ l_2 = 192\ bit\ ;\ l_3 = 256\ bit\}$
T_k, D_k : key update period for service request
k: periods for updating keys based on security level types
$\Delta T, \Delta D$: adjacent key updating period intervals

Fig 6a

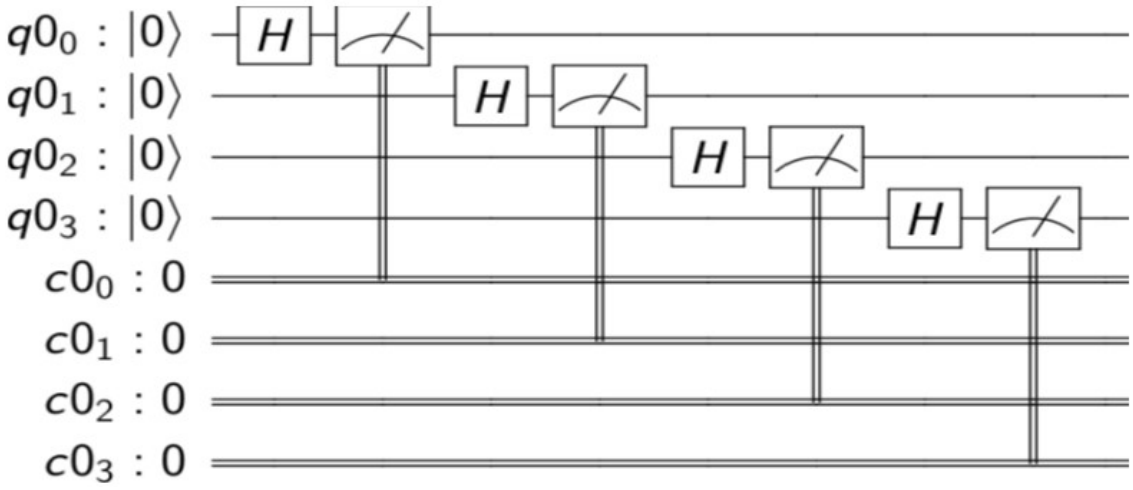


Fig 7a Prior Art

```
uint64_t state0 = 1;
uint64_t state1 = 2;
uint64_t xorshift128plus() {
  uint64_t s1 = state0;
  uint64_t s0 = state1;
  state0 = s0;
  s1 ^= s1 << 23;
  s1 ^= s1 >> 17;
  s1 ^= s0;
  s1 ^= s0 >> 26;
  state1 = s1;
  return state0 + state1;
}
```

Fig 7b Prior Art

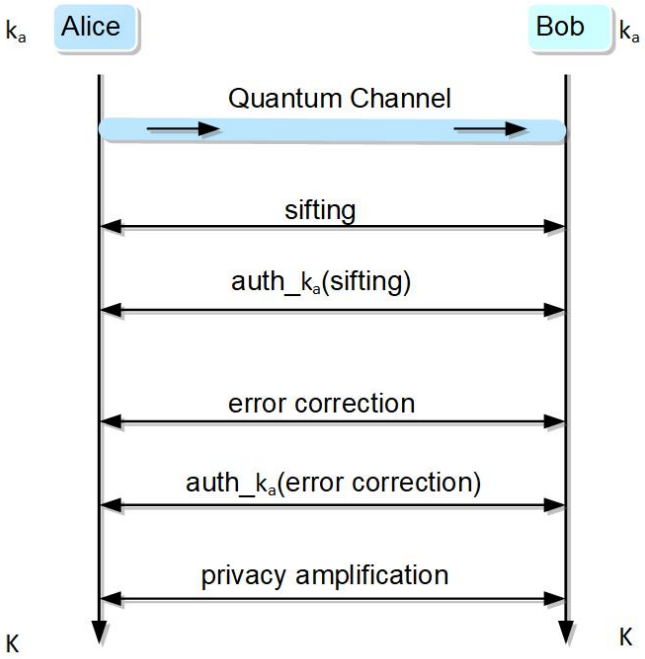


Fig 8 Prior Art